Random Variables, an AMPL Extension for Stochastic Programming

David M. Gay

Optimization and Uncertainty Estimation

dmgay@sandia.gov

+1-505-284-1456

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under contract DE-AC04-94AL85000.
Released as SAND2006-4717C.

AMPL summary

AMPL: a language for *mathematical programming problems:*

$$\text{minimize } f(x)$$
$$\text{s.t. } \ell \le c(x) \le u,$$

with $x \in \Re^n$ and $c : \Re^n \to \Re^m$ given algebraically and some x_i discrete.

AMPL goals

- Easy transcription from math (*avoid mistakes*)
- Explicit indexing (*no hidden magic*)
- Declare before use (*one-pass reading*)
- Separate model, data, commands (*orthogonality*)
- Separate solvers (*open solver interface*)
- Update entities as needed (*lazy evaluation*)
- Builtin math. prog. stuff (*presolve, reduced costs*)
- Aim for large scale nonlinear (*sparsity, generality*)

Example model: dieti.mod

```
set NUTR;  set FOOD;
param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];
param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];
param amt {NUTR,FOOD} >= 0;
var Buy {j in FOOD} integer >= f_min[j], <= f_max[j];
minimize Total_Cost:
            sum {j in FOOD} cost[j] * Buy[j];
subject to Diet {i in NUTR}:
   n_min[i] <= sum {j in FOOD} amt[i,j] * Buy[j]
            <= n_max[i];
```

Example data file: diet2a.dat (beginning)

```
data;
set NUTR := A B1 B2 C NA CAL ;
set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR ;

param:   cost  f_min  f_max :=
  BEEF   3.19    2     10
  CHK    2.59    2     10
  FISH   2.29    2     10
  HAM    2.89    2     10
  MCH    1.89    2     10
  MTL    1.99    2     10
  SPG    1.99    2     10
  TUR    2.49    2     10   ;
```

Example data file continued: `diet2a.dat`

```
param:    n_min  n_max :=
   A        700  20000
   C        700  20000
   B1       700  20000
   B2       700  20000
   NA         0  50000
   CAL    16000  24000 ;

param amt (tr):
             A    C    B1   B2    NA   CAL :=
   BEEF     60   20   10   15    938   295
   CHK       8    0   20   20   2180   770
   FISH      8   10   15   10    945   440
   HAM      40   40   35   10    278   430
   MCH      15   35   15   15   1182   315
   MTL      70   30   15   15    896   400
   SPG      25   50   25   15   1329   370
   TUR      60   20   15   10   1397   450 ;
```

Example session

```
ampl: model dieti.mod; data diet2a.dat;
ampl: option solver scplex; solve;
CPLEX 10.0.1: optimal integer solution; objective 119.3
10 MIP simplex iterations; 1 branch-and-bound nodes
ampl: display Buy;
Buy [*] :=
BEEF   9
 CHK   2
FISH   2
 HAM   8
 MCH  10
 MTL  10
 SPG   7
 TUR   2
;
```

Stochastic Programming — Motivation

Data often not known exactly, e.g.,

- prices
- demands
- rainfall
- transit times
- interest rates
- inflation rates

Stochastic Programming Approaches

Approaches include

- Modifying objective: instead of minimizing $f(x)$,
    - minimize $E(f(x))$
    - minimize $E(f(x)) + \alpha Var(f(x))$
- Modifying constraints: instead of satisfying a constraint exactly,
    - satisfy with probability $1 - \epsilon$
    - fail to satisfy with probability ϵ

What's random?

Potentially random entities include

- lower and upper bounds on
  - variables
  - constraints
- coefficients, e.g.,
  - costs
  - returns
  - rates
- function arguments

AMPL extension: random variables

Debated whether to add "*random parameters*" or "*random variables*".

Internally, they act like nonlinear variables, and "random variable" is a conventional term, so `random` in a `var` declaration introduces a random variable:

```
var x random;
```

Declarations may specify a value (with `=` or `default`):

```
var y random = Uniform01();
```

or subsequently be assigned:

```
let x := Normal(0,2);
```

Dependent random variables

Dependent random variables may only be declared in `var` ... `=` and `var` ... `default` declarations:

```
var x random;

var y = x + 1;
```

Random variables may appear as variables in constraint and objective declarations:

```
s.t. Demand: sum {i in A} build[i] >= y;
```

Seeing random variable values

Printing commands see random variables as strings expressing distributions...

```
var x random = Normal01();
var y = x + Uniform(3,5);
display x, y;
```

gives

```
x = 'Normal01()'
y = 'Uniform(3, 5) + x'
```

Sampling random variables

```
display {1..5} (Sample(x), Sample(y));
```

gives

```
:    Sample(x) Sample(y)     :=
1     1.51898     3.62453
2    -3.65725     2.50557
3    -0.412257    5.4215
4     0.726723    2.89672
5    -0.606458    3.776
;
```

Conventional uses of random functions

Without random, we get ordinary sampling:

```
var x := Uniform(0,10);
minimize zot: (x - Normal01())^2;
display x;
expand zot;
```

gives

```
x = 6.09209

minimize zot:
        (x - 1.51898)^2;
```

New builtin functions

New “builtin” functions for solvers to interpret:

- `Expected`(ξ)
- `Moment`(ξ, n), $n = 1, 2, 3, \ldots$
- `Percentile`(ξ, p), $0 \leq p \leq 100$
- `Sample`(ξ)
- `StdDev`(ξ)
- `Variance`(ξ)
- `Probability`$(\mathit{logical\ condition})$

What happens when?

Stages indicate what happens when.

SMPS convention: Stage = event followed by decision, perhaps with first stage "event" known.

A variable is split into separate copies, one for each realization of its stage (but not of subsequent stages).

For more on SMPS, see

`http://myweb.dal.ca/gassmann/smps2.htm`

New “system suffix” .stage

New reserved suffix .stage, e.g.,

```
set A; set Stages;
var x {A, s in Stages} suffix stage s;
```

or

```
var x {A, s in Stages};
...
let {a in A, s in Stages}
        x[a,s].stage := s;
```

Example: stochastic diet problem

Buy in two stages; constrain budget in first stage,
suffer random price changes in second stage.
What to buy in first stage?

Old:

```
var Buy {j in FOOD} integer >= f_min[j],
     <= f_max[j];
```

New:

```
set T = 1 .. 2;     # times (stages)
var Buy {FOOD, t in T} integer >= 0
     suffix stage t;
s.t. FoodBounds {j in FOOD}: f_min[j]
     <= sum{t in T} Buy[j,t] <= f_max[j];
```

Stochastic diet problem (cont'd)

Old:

```
minimize Total_Cost:
      sum {j in FOOD} cost[j] * Buy[j];
```

New:

```
var CostAdj {FOOD} random;
minimize Total_Cost:
     sum {j in FOOD} cost[j] * Buy[j,1]
     + Expected(sum {j in FOOD}
       cost[j]*CostAdj[j]*Buy[j,2]);
```

Stochastic diet problem (cont'd)

Old:
```
sum {j in FOOD} amt[i,j] * Buy[j]
```

New:
```
sum {j in FOOD, t in T}
          amt[i,j] * Buy[j,t]
param init_budget;
s.t. Init_Bud: sum {j in FOOD} Buy[j,1]
          <= init_budget;
...
let{j in FOOD} CostAdj[j]
          := Uniform(.7, 1.3);
```

“Constant” distributions

Assign numerical value to random variable $\Longrightarrow$ simplified problem (for debugging and model development).

Example:

```
        let{j in FOOD} CostAdj[j]

              := Sample(Uniform(.7, 1.3));
```

With imported function `Expected`$(x) = x$, this works with conventional solvers.

Some things work now

Things that work include

- Most details of random-variable handling
  - Declarations
  - Assignments of distributions
  - Assignments of constants
  - Printing and sampling (in AMPL sessions)
  - Determining what the solver will see as linear
- Writing `.nl` files with random distributions
- Suffix "`.stage`" and functions of distributions.

Work in progress

Updates to solver-interface library (for sampling), sample drivers not yet finished. Planned drivers include

- Adjust `.nl` file readers to pose *deterministic equivalent.*
- Program to write `.nl` file for deterministic equivalent.
- Program to write SMPS format.
- Driver for Gassmann's `MSLiP`.
- Driver for importance-sampling code (Infanger's?).

Wish-list items

To do when possible...

- Constraint-programming extensions: variables in subscripts should help with handling scenarios.
- "Tree" sets (for scenarios).
- Ordered sets of $arity > 1$.
- Non-program entities involving variable values.
- Update *Hooking Your Solver to AMPL*.

For more details (`dmgay@sandia.gov` or `dmg@ampl.com`)

http://www.ampl.com points to

- The AMPL book
- examples (models, data)
- descriptions of new stuff (in book 2nd ed., not 1st)
- downloads
  - student binaries
  - solver interface library source
  - "standard" table handler & source
  - papers and reports